CIRCUITOS LÓGICOS

Relatório IV

Discente: Rebeca de Macêdo Ferreira. 2016000524

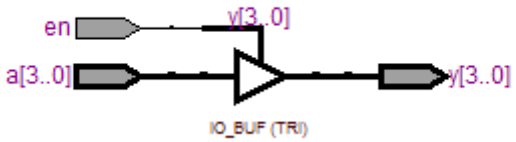
TRISTATE

Para as simulações desse projeto usamos o seguinte cogido em VHD que se encontra no livro Harris and Harris

```
1   library IEEE; use IEEE.STD_LOGIC_1164.all;
2
3   entity tristate is
4           port (a: in STD_LOGIC_VECTOR (3 downto 0);
5                 en: in STD_LOGIC;
6                 y: out STD_LOGIC_VECTOR (3 downto 0));
7   end;
8
9   architecture synth of tristate is
10
11  begin
12  y <= "ZZZZ" when en = '0' else a;
13  end;
```

As variáveis de entrada são a (4 bits) e em (1 bit), a variável de saída é o y (4 bits)

Compilamos o código no Quartus e após correção de alguns erros selecionamos Toos → Netlist Viewers → RTL Viewer para obter a visualização do circuito de portas lógicas.



Feito isto fomos fazer as simulações RTL level e Gate level com o Model-Sim, para isso precisamos do test bench em VHD e de um arquivo .TV com as informações da tabela verdade pra esse projeto.

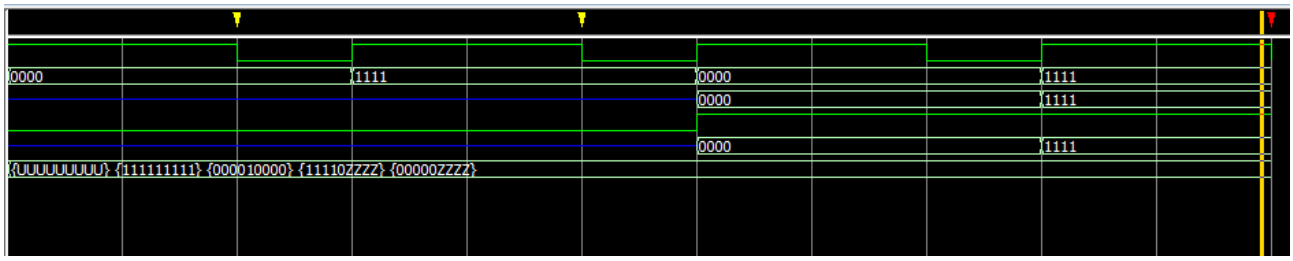
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use STD.TEXTIO.ALL ;

entity testbench_tristate is
    -- no inputs or outputs
end;

architecture sim of testbench_tristate is
    component tristate
        port (a: in STD_LOGIC_VECTOR(3 downto 0);
              en: in STD_LOGIC;
              y: out STD_LOGIC_VECTOR(3 downto 0));
    end component;

signal clk: STD_LOGIC;
signal a, y: STD_LOGIC_VECTOR(3 downto 0);
signal en: STD_LOGIC;
signal yexpected: STD_LOGIC_VECTOR(3 downto 0);
constant MEMSIZE: integer := 4;
type tvarray is array (MEMSIZE downto 0) of STD_LOGIC_VECTOR (8 downto 0);
signal testvectors: tvarray;
shared variable vectornum, errors: integer;
begin
-- instantiate device under test
dut: tristate port map (a, en, y);
-- generate clock
process begin
    clk <= '1'; wait for 10 ns;
    clk <= '0'; wait for 5 ns;
end process;
-- at start of test, load vectors
```

testbench tristate

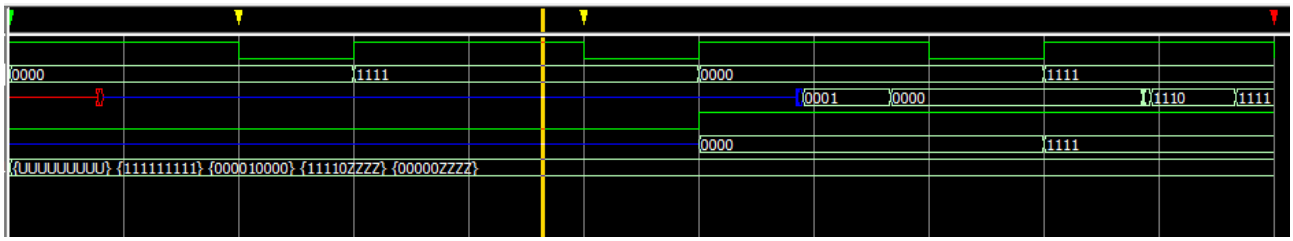
```
00000_ZZZZ
11110_ZZZZ
00001_0000
11111_1111
```

Assim adicionamos esse arquivo testbench.vhd no projeto e a tabela verdade na pasta Simulation → ModelSim.

Para simular o RTL level selecionamos Tools → Run EDA Simulation Tool → RTL Level e obtivemos o seguinte



Visto que não haviam erros no RTL level, fomos simular o Gate level em Tools → Run EDA Simulation Tool → Gate Level



como foi proposto na preparação de laboratório, o testbench desse projeto foi uma adaptação de um testbench já usado anteriormente, então como os clocks já estavam com o atraso ajustado não obtivemos erros, porém caso não tivessem ajustado seria necessário fazer as alterações para deixá-los corretos.

Visualização em lista da simulação:

List - Default

ps delta	/testbench_tristate/a /testbench_tristate/y /testbench_tristate/en /testbench_tristate/yexpected
0 +0	UUUU UUUU U UUUU
0 +2	0000 XXXX 0 ZZZZ
3821 +2	0000 XXZX 0 ZZZZ
3831 +2	0000 XZZX 0 ZZZZ
4102 +2	0000 ZZZZ 0 ZZZZ
15000 +2	1111 ZZZZ 0 ZZZZ
30000 +2	0000 ZZZZ 1 0000
34232 +1	0000 ZZ0Z 1 0000
34242 +1	0000 Z00Z 1 0000
34513 +1	0000 0001 1 0000
38289 +1	0000 0000 1 0000
45000 +2	1111 0000 1 1111
49285 +1	1111 0010 1 1111
49302 +1	1111 0110 1 1111
49609 +1	1111 1110 1 1111
53289 +1	1111 1111 1 1111

MUX 4 ESTRUTURAL

Para a simulação desse projeto usamos também um código em VHD para o multiplex de 4 entradas estrutural, encontrado no livro Harris and Harris, com as informações pertinentes. Para fazer esse Multiplex faz se necessário o uso do projeto de mux de 2 entradas, pois o usaremos como “função” para construir o mux de 4 entradas estrutural. Colocamos então o código VHD do mux de 2 entradas na pasta deste projeto.

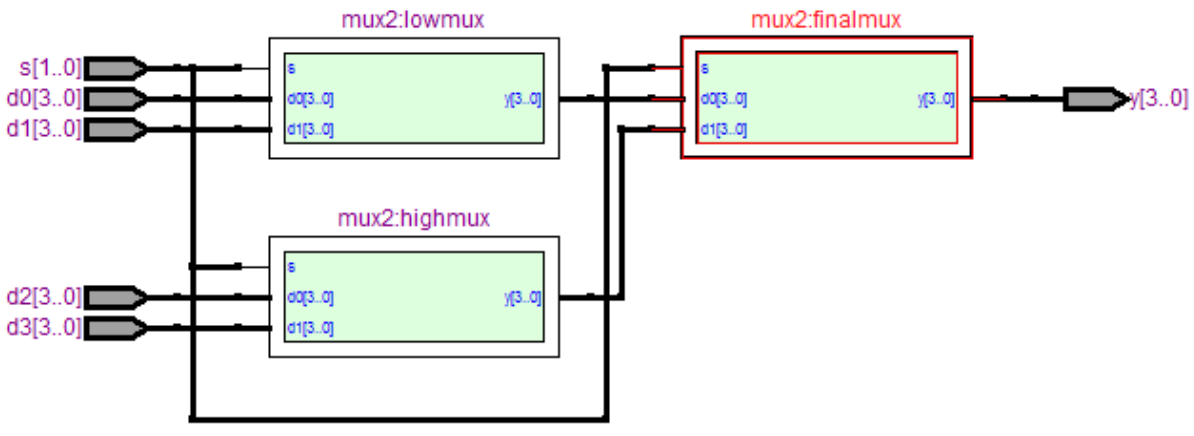
```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4_estrutural is
    port (d0, d1,
          d2, d3: in STD_LOGIC_VECTOR (3 downto 0);
          s: in STD_LOGIC_VECTOR (1 downto 0);
          y: out STD_LOGIC_VECTOR (3 downto 0));
end;

architecture struct of mux4_estrutural is
component mux2
    port (d0, d1: in STD_LOGIC_VECTOR (3 downto 0);
          s: in STD_LOGIC;
          y: out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal low, high: STD_LOGIC_VECTOR (3 downto 0);

begin
lowmux: mux2 port map (d0, d1, s(0), low);
highmux: mux2 port map (d2, d3, s(0), high);
finalmux: mux2 port map (low, high, s(1), y);
end;
```

código VHD do mux 4 estrutural onde o mux2 faz papel de uma função

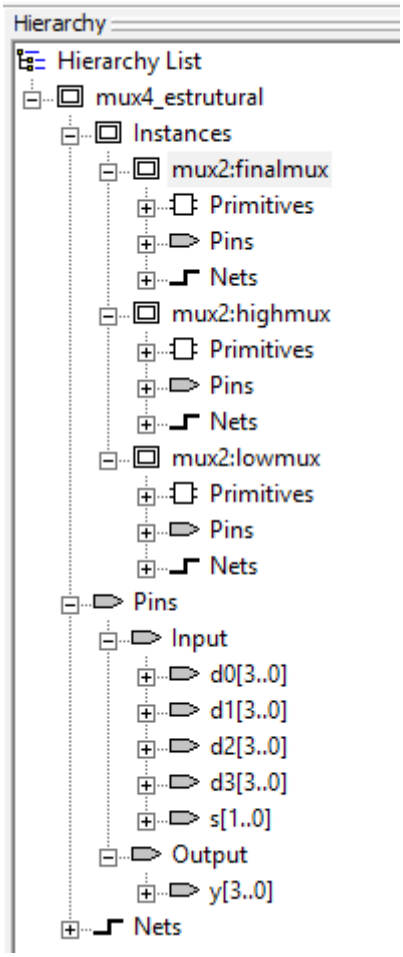
```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
    port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
         s: in STD_LOGIC;
         Y: OUT STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
    y <= d0 when s = '0' else d1;
end;
```

código VHD do mux2

Compilamos o arquivo no Quartus e selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito de portas lógicas do projeto



Lista de hierarquia do mux de 4 entradas estrutural



Para fazer simulação com o ModelSim é preciso do testbench e do arquivo .tv com as informações da tabela verdade para este projeto

```
7   entity testbench_mux4 is
8       -- no inputs or outputs
9   end;
10
11  architecture sim of testbench_mux4 is
12      component mux4_estrutural
13          port (d0, d1, d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
14                s: in STD_LOGIC_VECTOR(1 downto 0);
15                y: out STD_LOGIC_VECTOR(3 downto 0));
16      end component;
17
18  signal clk: STD_LOGIC;
19  signal d0, d1, d2, d3, y: STD_LOGIC_VECTOR(3 downto 0);
20  signal s: STD_LOGIC_VECTOR(1 downto 0);
21  signal yexpected: STD_LOGIC_VECTOR(3 downto 0);
22  constant MEMSIZE: integer := 8;
23  type tvarray is array (MEMSIZE downto 0) of STD_LOGIC_VECTOR (21 downto 0);
24  signal testvectors: tvarray;
25  shared variable vectornum, errors: integer;
26  begin
27  -- instantiate device under test
28  dut: mux4_estrutural port map (d0, d1, d2, d3, s, y);
29  -- generate clock
30  process begin
31      clk <= '1'; wait for 15 ns;
32      clk <= '0'; wait for 15 ns;
33  end process;
```

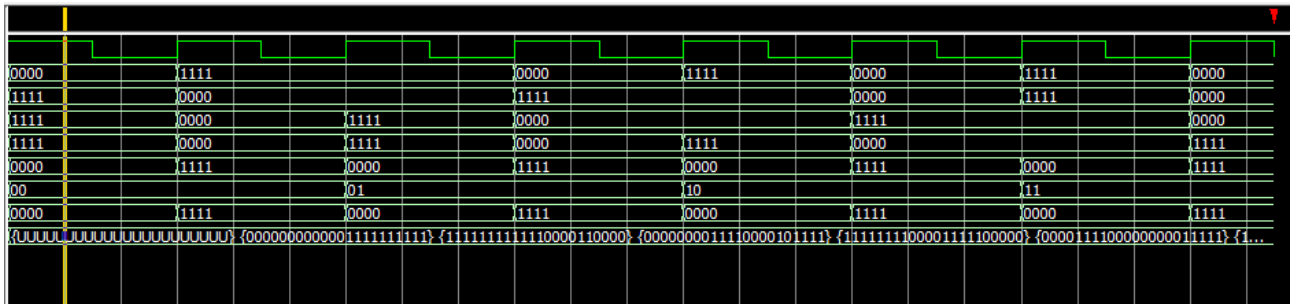
testbench.vhd

Novamente o testbench usado foi uma adaptação de projetos anteriores, como sugerido, então os clocks não foram alterados.

```
1  00001111111111100_0000
2  11110000000000000_1111
3  11110000111111101_0000
4  00001111000000001_1111
5  11111110000111110_0000
6  00000001111000010_1111
7  11111111111000011_0000
8  00000000000111111_1111
```

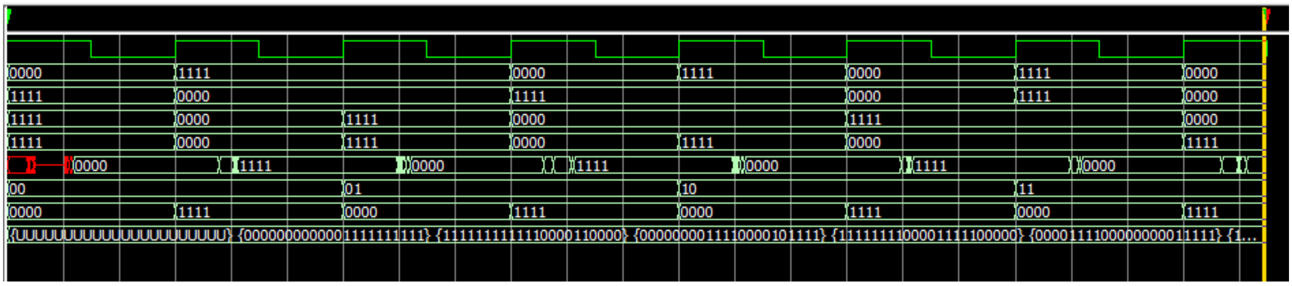
Arquivo .TV utilizado

Para simular o RTL level selecionamos Tools → Run EDA Simulation Tool → RTL Level



```
Transcript
# Loading work.testbench_mux4(sim)
# Loading work.mux4_estrutural(struct)
# Loading work.mux2(synth)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 225 ns  Iteration: 1  Process: /testbench_mux4/line__76 File: C:/Users/aluno/Desktop/circuitos/mux4_estrutural/testbench/testbench_mux4.vhd
# Break in Process line__76 at C:/Users/aluno/Desktop/circuitos/mux4_estrutural/testbench/testbench_mux4.vhd line 92
```

Em seguida simulamos o Gate Level em Tools → Run EDA Simulation Tool → Gate Level




```
Transcript
# Loading instances from mux4_estrutural_vhd.sdo
# Loading timing data from mux4_estrutural_vhd.sdo
# ** Note: (vsim-3587) SDF Backannotation Successfully Completed.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_mux4 File: C:/Users/aluno/Desktop/circuitos/mux4_estrutural/testbench/testbench_mux4.vhd
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 225 ns  Iteration: 1  Process: /testbench_mux4/line__76 File: C:/Users/aluno/Desktop/circuitos/mux4_estrutural/testbench/testbench_mux4.vhd
# Break in Process line  76 at C:/Users/aluno/Desktop/circuitos/mux4 estrutural/testbench/testbench mux4.vhd line 92
```

Como o clock já está com o atraso ajustado não obtivemos erros, caso não estivessem teríamos que ajustá-lo.

MUX DE 2 ENTRADAS ESTRUTURAL

Para esse projeto usamos o código VHD encontrado no livro Harris and Harris. Aqui usamos o Tristate como uma função do projeto, além de uma função ‘inversor’ (por questões de compatibilidade), portanto fez se necessário colocar o código VHD do tristate e do inversor no diretório deste projeto.

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inversor is

port (a: in STD_LOGIC;
    y: out STD_LOGIC);
end;

architecture synth of inversor is
begin
y <= not a;
end;
```

Código VHD do inversor

mux2_estrutural.vhd | tristate.vhd

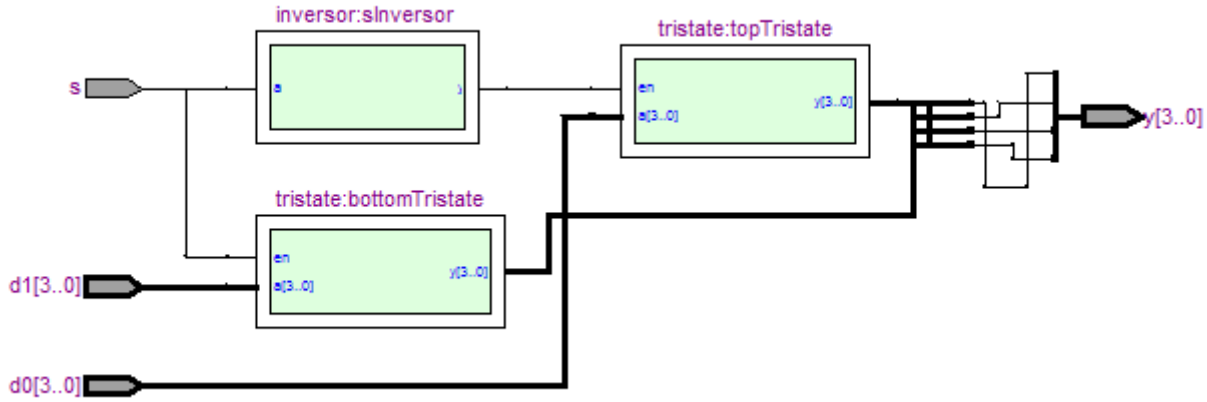
```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity tristate is
        port (a: in STD_LOGIC_VECTOR (3 downto 0);
            en: in STD_LOGIC;
            y: out STD_LOGIC_VECTOR (3 downto 0));
end;

architecture synth of tristate is

begin
y <= "ZZZZ" when en = '0' else a;
end;
```

código VHD tristate

```vhdl
library IEEE;

use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity mux2_estrutural is
port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
    s: in STD_LOGIC;
    y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2_estrutural is

component tristate
port(a: in STD_LOGIC_VECTOR(3 downto 0);
en: in STD_LOGIC;
y: out STD_LOGIC_VECTOR(3 downto 0));
end component;

component inversor
port (a: in STD_LOGIC;
     y: out STD_LOGIC);
end component;

signal notS: STD_LOGIC;

begin
    sInversor: inversor port map(s, notS);
    topTristate: tristate port map(d0, notS, y);
    bottomTristate: tristate port map(d1, s, y);

end;
```

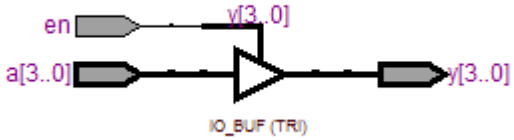
código VHD do mux 2 estrutural

Compilamos os arquivos no Quartus e selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito de portas lógicas do projeto

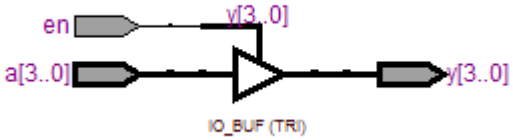


Desse modo verifica-se que o Mux foi dividido em 3 partes, são elas 1 inversos e 2 tristates.

Inversor



Tristate



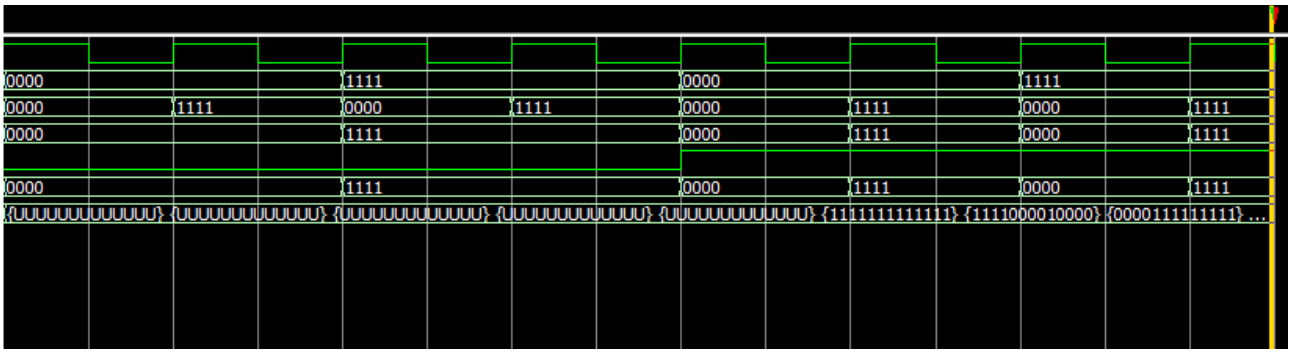
Tristate

Em seguida adicionamos ao projeto o código testbench em VHD e o arquivo .TV com informações da tabela verdade do projeto, novamente foi reutilizado um testbench anterior, já com os clocks ajustados.

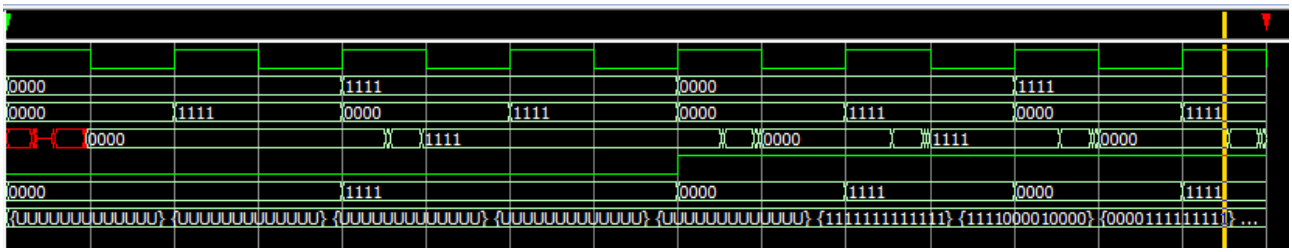
```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_arith.ALL;
4   use IEEE.STD_LOGIC_unsigned.ALL;
5   use STD.TEXTIO.ALL ;
6
7   entity testbench_mux2 is
8       -- no inputs or outputs
9   end;
10
11  architecture sim of testbench_mux2 is
12      component mux2_estrutural
13          port (d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
14                s: in STD_LOGIC;
15                y: out STD_LOGIC_VECTOR(3 downto 0));
16      end component;
17
18  signal clk: STD_LOGIC;
19  signal d0, d1, y: STD_LOGIC_VECTOR(3 downto 0);
20  signal s: STD_LOGIC;
21  signal yexpected: STD_LOGIC_VECTOR(3 downto 0);
22  constant MEMSIZE: integer := 12; --4
23  type tvarray is array (MEMSIZE downto 0) of
24  STD_LOGIC_VECTOR (12 downto 0);
25  signal testvectors: tvarray;
26  shared variable vectornum, errors: integer;
27  begin
28  -- instantiate device under test
29  dut: mux2_estrutural port map (d0, d1, s, y);
30  -- generate clock
31  process begin
32      clk <= '1'; wait for 10 ns;
33      clk <= '0'; wait for 10 ns;
```

Testbench mux 2 estrutural

Para simular o RTL level selecionamos Tools → Run EDA Simulation Tool → RTL Level

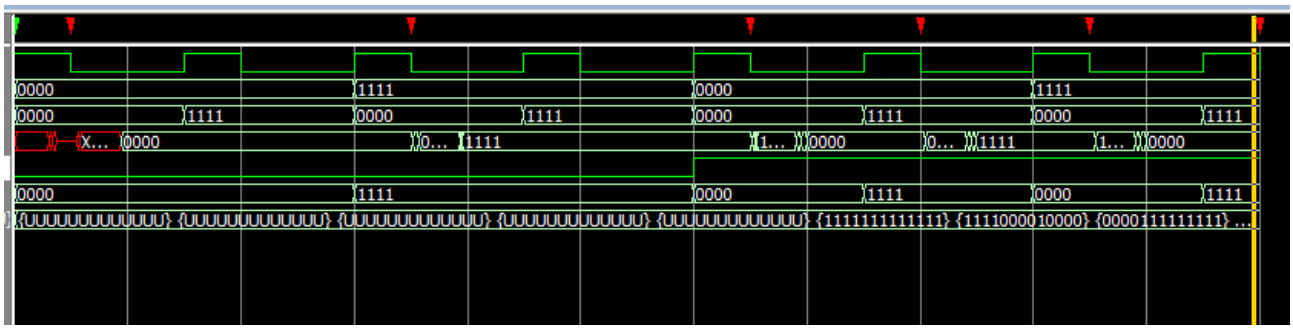


e para simular o Gate level selecionamos Tools → Run EDA Simulation Tool → Gate Level



Caso os clocks fossem alterados (desajustados), para 5ns, por exemplo, teríamos erros no Gate level…

```
process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 10 ns;
end process;
-- at start of test, load vectors
```



```
Transcript
# ** Error: Vetor deu erro n. Teste: 7. Esperado yesp ='1'Valor Obtido: y(0) ='0'
#    Time: 110 ns  Iteration: 1  Instance: /testbench_mux2
# ** Error: Vetor deu erro n. Teste: 7. Esperado yesp ='1'Valor Obtido: y(1) ='0'
#    Time: 110 ns  Iteration: 1  Instance: /testbench_mux2
# ** Error: Vetor deu erro n. Teste: 7. Esperado yesp ='1'Valor Obtido: y(2) ='0'
#    Time: 110 ns  Iteration: 1  Instance: /testbench_mux2
# ** Error: Vetor deu erro n. Teste: 7. Esperado yesp ='1'Valor Obtido: y(3) ='0'
#    Time: 110 ns  Iteration: 1  Instance: /testbench_mux2
# ** Failure: 8tests completed, errors = 24
#    Time: 110 ns  Iteration: 1  Process: /testbench_mux2/line__75 File: C:/Users/aluno/Desktop/circuit
# Break in Process line__75 at C:/Users/aluno/Desktop/circuitos/mux2_estrutural/mux2_estrutural/testben
```

Visualização em lista:

List - Default

ps delta	/testbench_mux2/d0 /testbench_mux2/d1 /testbench_mux2/y /testbench_mux2/s /testbench_mux2/yexpected
0 +0	UUUU UUUU UUUU U UUUU
0 +2	0000 0000 XXXX 0 0000
0 +4	0000 0000 UUUU 0 0000
3112 +1	0000 0000 UUXU 0 0000
3388 +1	0000 0000 UUXX 0 0000
3663 +1	0000 0000 XUXX 0 0000
3689 +1	0000 0000 XXXX 0 0000
5473 +1	0000 0000 XX0X 0 0000
5768 +1	0000 0000 XX00 0 0000
9412 +1	0000 0000 0X00 0 0000
9462 +1	0000 0000 0000 0 0000
20000 +2	0000 1111 0000 0 0000
40000 +2	1111 0000 0000 0 1111
45191 +1	1111 0000 0010 0 1111
45768 +1	1111 0000 0011 0 1111
49412 +1	1111 0000 1011 0 1111
49462 +1	1111 0000 1111 0 1111
60000 +2	1111 1111 1111 0 1111

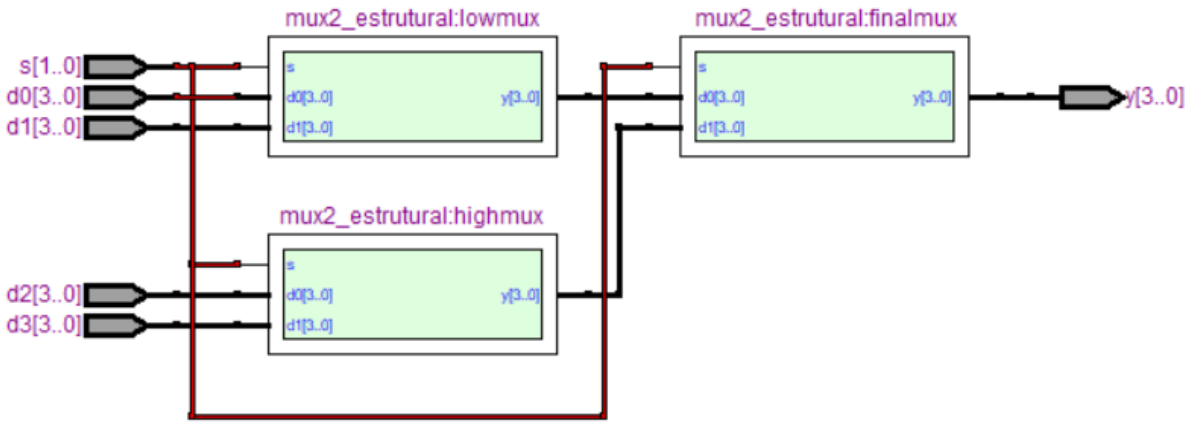
MUX 4 ESTRUTURAL ALTERADO

Alteramos o Mux de 4 entradas estrutural, e ao invés de usarmos o mux de 2 entradas simples, usamos o mux de 2 entradas estrutural como uma função deste projeto, além disso utilizamos também o inversor, tristate e, claro, o próprio mux de 4 entradas estrutural. Logo, no diretório deste projeto foi inserido os códigos em VHD do inversor, tristate, mux 4 estrutural e mux 2 estrutural.

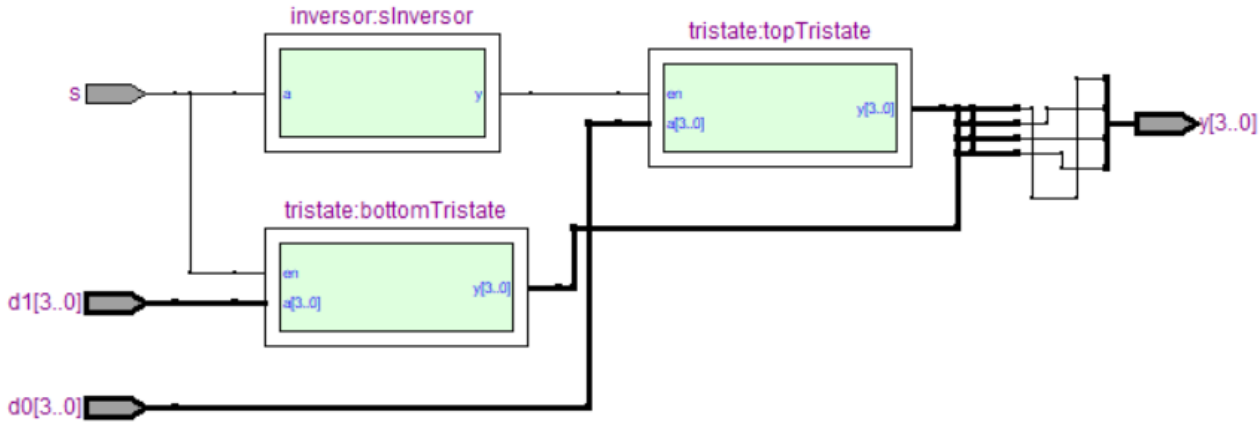
```
1  library IEEE; use IEEE.STD_LOGIC_1164.all;
2
3  entity mux4_estrutural2 is
4      port (d0, d1,
5            d2, d3: in STD_LOGIC_VECTOR (3 downto 0);
6            s: in STD_LOGIC_VECTOR (1 downto 0);
7            y: out STD_LOGIC_VECTOR (3 downto 0));
8  end;
9
10 architecture struct of mux4_estrutural2 is
11 component mux2_estrutural
12     port (d0, d1: in STD_LOGIC_VECTOR (3 downto 0);
13           s: in STD_LOGIC;
14           y: out STD_LOGIC_VECTOR (3 downto 0));
15 end component;
16 signal low, high: STD_LOGIC_VECTOR (3 downto 0);
17
18 begin
19 lowmux: mux2_estrutural port map (d0, d1, s(0), low);
20 highmux: mux2_estrutural port map (d2, d3, s(0), high);
21 finalmux: mux2_estrutural port map (low, high, s(1), y);
22 end;
```

Código VHD mux4 estrutural alterado

Selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito em portas logicas.

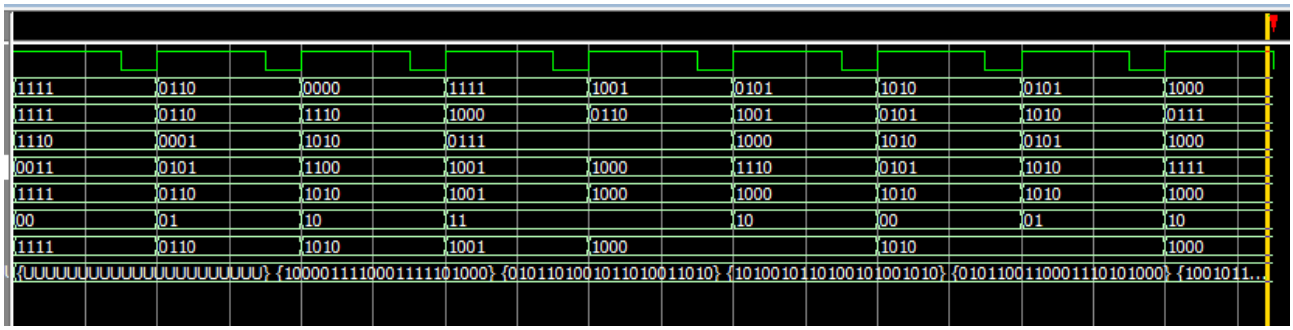


Cada instancia “mux2_estrutural...” tem em si o inversor e seus dois tristates, como foi observado nas simulações do mux 2 estrutural…



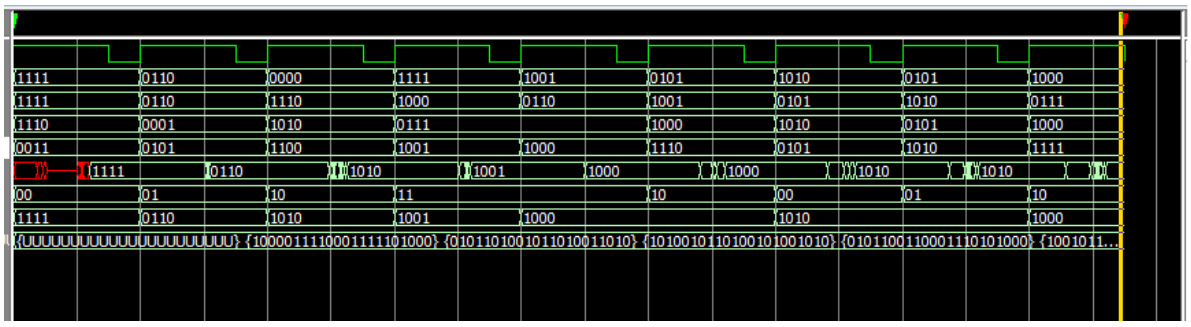
O testbench usado foi o mesmo do mux 4 estrutural alterado, apenas com algumas alterações, e o arquivo .TV utilizado foi o do próprio mux 4 estrutural.

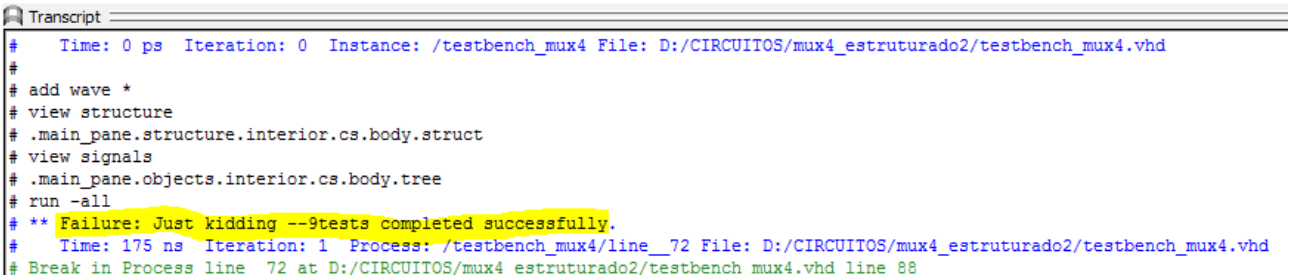
Simulamos o RTL level em Tools → Run EDA Simulation Tool → RTL Level



```
Transcript
# Loading work.tristate(synth)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --9tests completed successfully.
#    Time: 175 ns  Iteration: 1  Process: /testbench_mux4/line__72 File: D:/CIRCUITOS/mux4_estruturado2/testbench_mux4.vhd
# Break in Process line__72 at D:/CIRCUITOS/mux4_estruturado2/testbench_mux4.vhd line 88
```

e simulamos o Gate Level em Tools → Run EDA Simulation Tool → Gate Level





ARVORE DE DIRETÓRIOS

Tristate

Aquivo com **código VHD**: tristate → tristate.vhd.

Arquivo com **testbench VHD:** tristate → testbench → testbench_tristate.vhd.

Arquivo com **tabela verdade .TV**: tristate → simulation → modelsim → tristate.tv.

Mux 4 entradas estutural

Aquivo com **código VHD** (mux2, mux4_estrutural): mux4_estrutural → mux2.vhd / mux4_estrutural.vhd.

Arquivo com **testbench VHD:** mux4_estrutural → testbench → testbench_mux4.

Arquivo com **tabela verdade .TV**: mux4_estrutural → simulation → modelsim → mux4.tv.

Mux 2 entradas estrutural

Aquivo com **código VHD** (inversor, mux2 estrut., tristate): mux2_estrutural → inversor.vhd / mux2_estrutural.vhd / tristate.vhd.

Arquivo com **testbench VHD:** mux2_estrutural → testbench → testbench_mux2.vhd.

Arquivo com **tabela verdade .TV**: mux2_estrutural → simulation → modelsim → mux2.tv.

Mux 4 entradas estrutural alterado

Aquivo com **código VHD** (inversor, mux2 estrut. tristate, mux4 estrut): mux4_estrutural2 → inversor.vhd / mux2_estrutural.vhd / mux4_estrutural.vhd / tristate.vhd.

Arquivo com **testbench VHD:** mux4_estrutural2 → testbench → testbench_mux4.

Arquivo com **tabela verdade .TV**: mux4_estrutural2 → simulation → modelsim → mux4.tv.